EXPLORING MAINSTREAM NATURAL LANGUAGE PROCESSING TECHNIQUES AND THEIR APPLICATION TO COMPUTATIONAL HUMOR

An Undergraduate Research Scholars Thesis

by

DHANANJAY KHANNA

Submitted to the Undergraduate Research Scholars program at
Texas A&M University
in partial fulfillment of the requirements for the designation as an

UNDERGRADUATE RESEARCH SCHOLAR

Approved by Research Advisor: Dr. Dylan Shell

May 2018

Major: Computer Science

TABLE OF CONTENTS

ABSTRACT

Exploring Mainstream Natural Language Processing Techniques and Their Application to Computational Humor

Dhananjay Khanna
Department of Computer Science and Engineering
Texas A&M University

Research Advisor: Dr. Dylan Shell
Department of Computer Science and Engineering
Texas A&M University

The purpose of this study is to document the creation of a software prototype that attempts to generate fact(s) about a variety of subjects in a legible English language format. Said prototype builds on top of an existing Natural Language Generation (NLG) research project called SimpleNLG. The format selected is that of the Harper's Index, which is a list of facts in a rigid format published monthly by Harper's Magazine. The rigidity of the format makes it easier to generate than regular language. This combined with the socio-economic and political topics that Harper's authors usually touch upon makes it ideal for the study of Computational Humor, since it forces the reader to wrestle with their understanding of the world, a skill necessary to use and perceive humor. It is our hope that in creating this prototype, we can further the knowledge that goes into creating intelligent and adaptable computers that serve far more day-to-day applications in our increasingly technological world.

ACKNOWLEDGEMENTS

I would like to thank my Faculty Advisor Dr. Dylan Shell for giving me the opportunity to work with him and for trusting me to pursue this research. I thank him for helping me grow intellectually and personally as a result of his mentorship over the past two years.

I also want to thank my Honors program Director Dr. Jennifer Welch, and Dr. Aakash Tyagi for their support throughout the duration of my undergraduate career.

I thank the staff and faculty of the Department of Computer Science and Engineering for providing the academic and financial support without which I would not have been able to complete my undergraduate education.

Finally, I thank my parents for their unconditional support of my efforts.

NOMENCLATURE

AI	Artificial Intelligence
NLP	Natural Language Processing
NLG	Natural Language Generation
GPSG	Generalized Phrase Structure Grammar

CHAPTER I

INTRODUCTION

Natural Language Processing is the field of Artificial Intelligence that concerns communication between a computer and a human through the medium of human language. Although NLP is increasingly prevalent in our lives, its sub-field of Computational Humor is still in its infancy. Little real progress has been made beyond a multitude of pun generating agents. Part of the issue lies in the complexities of human language, which involves ambiguity and abstraction. On top of that, the study of the cognitive resources that allow human beings to perceive humor is limited. This means that the only progress in the field has been limited to superficial, wordplay based jokes. Here's an example:

Q: How many eggs did Lance Armstrong buy?

A: A 'Biker's' dozen

There isn't any depth to this joke. The AI recognized that 'biker' and 'baker' sound similar, and then picked Lance Armstrong, as he is a professional bicyclist.

The issue we want to address is how can a computer generate similar short phrases, but actually include real data and some contextual understanding of the real world.

This project was thus inspired by a simple question:

What if computers could compose small, pithy, through-provoking summaries of complex data?

The aim of this project is to build an Intelligent Agent capable of producing a specific type of content called a Harper's Index, published monthly by Harper's Magazine. Each Harper's Index is a collection of facts in a specific 'Phrase: Number' format, that seek to comment on the current state of social, political and economic affairs. While there are unlimited

possible types of content that can approach current affairs, we chose the Harper's Index because it has a rigid format, making it easier to produce than an average paragraph.

We do want to clarify that when we use the term 'fact', we are using it in the loosest sense of the word. We would like to define that a fact (for our purposes) is a self-contained description of the relationship between parties involved. We are not making any claim that every Harper's Index we produce is factually correct in our reality; instead each fact (Harper's Index) we create is simply an expression of a relationship we ourselves have defined, with no guarantee to its correctness in the real world. Doing so allows us to play around with different types of content and achieve the best possible summaries of varying types of knowledge.

We aim to use our findings to further research into Computational Humor. Our initial challenge is generation: the act of building an agent capable of producing even random text in this format. We will then fine-tune said agent to produce text that reads just like a human-written Harper's Index. In doing so, we hope to identify what factors separate superficial content where only the words matter (like a pun or a weather report) from content with deeper context, that requires delayed reasoning and an understanding of the world (like real jokes). The motivation here is to explore how to build agents that are capable of interacting like human beings, thus incorporating them into our day-to-day lives further than what is currently possible.

Let us look at an example of a Harper's Index:

Percentage increase since last year in the number of U.S. taxpayers who fear an IRS audit : 64
Number of successive years that the IRS has audited fewer people than it did the year before: 7

This example draws light to the stark contrast between the public image of the IRS and the reality of the IRS. It may also lead one to wonder the reasons behind this contrast; one possible explanation is the rise of politicking against 'establishment' institutions. Or it may lead one to question the outcome: perhaps the newfound fear of the IRS reduces crimes and thus the need for the IRS to audit taxpayers, making one wonder if this was the IRS' aim all along. Another great example:

Percentage of Republicans living within 350 miles of Mexico who oppose the construction of a border wall: 34
Percentage of Republicans living farther than 350 miles from Mexico who do: 21

This example is simpler: it picks an issue that is prevalent in the news, and makes one question what reasoning could those who live closer to the US-Mexico border have to be more averse to the border wall than others who share very similar beliefs but live geographically farther.

Harper's indices often work in pairs, with the aim being to bring out a contrast between facts and public perception. However, they can often get a point across in a single statement. For instance:

Factor by which white girls in the US are more likely than black girls to binge drink : 2

Again, the mind wonders; why is this the case? It could be related to vastly different income levels between average white and black families, and correspondingly the amount of money made available to young adult women in the form of allowances. It could also be related to the racial prejudice of those indulging themselves in the act of binge drinking. Young black women may be more fearful of the consequences of such action if they come in contact with police officers than young white women who expect they will be respected and treated fairly. These are just two of many possible explanations.

As evidenced by the above examples, Harper's Indices set off a chain reaction of reasoning, given a minimal set of information about the real world. These examples showcase the fact that the humans writing these indices write them with the aim of forcing people to wrestle with their understanding of the world, a skill necessary to use and perceive humor. This combined with the rigid format of the Harper's index makes it an easier jumping off point for studying Natural Language Generation than trying to produce entire paragraphs from scratch.

The purpose of this research goes much farther than simply building an AI capable of expressing more complex humor. Its aim is to improve the way in which we consume information about the increasingly complex world around us. As our world has become more complex, the average citizenry turns to sources of information that are over-simplified and packaged for mass consumption. This has been seen as a negative sign, with some critics claiming that the very existence of democracy leads to dumbing down of public discourse. This author contends that the medium of information must keep up with the people than the other way around. This period in time is hardly the first where complex issues have had to be broken down for the masses; for instance, President John F. Kennedy did an excellent job of breaking down the complexity of the Moon landing operation all the way back in the nineteen-sixties, from the

scientific particulars of the mission itself to the difficult process of legislative and bureaucratic apportionment that made the mission possible.

In an age where we consume the news in the form of fifteen-second sound bites and a hundred and forty character tweets, our hope is that the success of our research can showcase that it is perfectly possible to disseminate far more complex information under similar rigid constraints of format and length, and that humor and wit can aid in that process.

CHAPTER II

METHODS

We present again the research question that we aim to answer:

What if computers could compose small, pithy, through-provoking summaries of complex data?

Specifically, we want to build an agent that is capable of the following:

1) Detecting and using quantitative properties of subjects
2) Following a grammatical template
3) Possibly using contrast to further the quality of content

A simple glance is enough to prove that the average Harper's Index meets all three requirements. Therefore, if we can build a prototype capable of producing Harper's Indices that read like a human wrote them, we will have succeeded.

Our methodology for this prototype is one of iterative design. As shown in Figure 1, we shall start our project in the opposite direction from past humor researchers; we will start from a very basic text generation agent, then iterate multiple times over the course of the academic year to come closer to the ideal agent by adding layers of representation to build the 'Miniverse' of objects and relations in which our agent attempts to find humor.

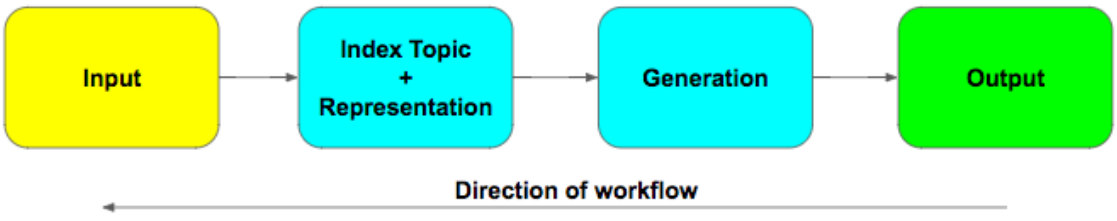

Figure 1. Expected AI Design compared to reverse engineering workflow plan. Generation refers to production of text; Representation refers to formatting and semantics.

It is important to define the two steps of the process to generate natural language.

Representation refers to building an agent's sense of the world and how various subjects are related to each other. This is usually done in the form of building a 'Knowledge Base' that contains some logical definition of various subjects and their relationships. That knowledge base can then be queried for isolating the subjects and relationship(s) we want to exploit.

Generation is the process of expressing those relationships by creating legible phrases and sentences by the agent. To put it simply, this is the part where the agent spurns out text that describes the relationship isolated above in a manner that can be reasonably interpreted and understood by a human observer.

It is important to specify why we are moving in the opposite direction than previous researchers in this field. Past researchers have gotten so caught up in the complexities of real world relationships that by the time they get to the part where the computer actually generates any text, there are severe constraints keeping the AI from being expressive and flexible in the content it generates about those objects and relationships. Our hope was to make the generator first in a way that it had the flexibility of inflection regardless of the set of information being passed in, resulting in richer, more verbose content.

Computational Humor then, is a two-step problem that involves:

1. Identifying those relationships that provide the fodder for humor in the real world.
2. Identifying the right syntax and semantics that actually get the joke across.

We aim to start with a study of the historical efforts made in the field of computational humor, as well as trying to understand the reasons that some of the leading figures of the field of AI are spending time and resources on it. It goes beyond the craze around chat-bots that firms like Apple pioneered with Siri, to make their bots more conversational. Humor is the natural

response of human beings to put a healthy distance between themselves and their circumstances (Stock and Strapparava 63). A simple computer is a fast, dumb machine that is only useful when told precisely what to do. For us to someday build agents that can be more useful across day-to-day applications, humor is an untapped avenue to studying contextual and semantic learning.

Let us now dive into the methodology that we shall employ to address the two logical halves of our project, and how they will be linked together to achieve our goal.

Generation

We present the first research sub-question we aim to answer:

Assuming we are given varying sets of subjects and corresponding quantitative facts about each subject, can we build a program capable of producing a Harper's Index for each subject, regardless of the nature of the subject and/or the facts we are aiming to generate?

If yes, can we hypothesize a grammar for the average computer generated Harper's Index?

To answer this research question, we must build a JAVA program that can generate a human readable Harper's Index if we 'hardcode' (pre-set) in information about the chosen subject. We will then attempt to hypothesize a Generalized Phrase Structure Grammar (GPSG) for a Harper's Index.

A GPSG is a theoretical framework used to define the syntax and semantics of a given piece of generated text. In simpler terms, a GPSG is a grammatical template.

We will aim to isolate the minimum required data points needed to fill the GPSG we hypothesize for a Harper's Index. For us to do this, we must first build our program and try to run it with a variety of subjects and corresponding numerical facts, and keep tweaking the generation code until we arrive at a consistent format that produces grammatically correct Harper's Indices regardless of the nature of the subject(s) we are trying to describe. Essentially,

we want to get as close to a piece of code that feels like a fill in the blanks exercise, where we just need to give the program a few major details, and it constructs the Harper's Index for us. However, the library is such that this won't be entirely possible, as the programmer still has to define inflections for a seemingly human-written Harper's Index. Once we have such a program written, that resulting code will be used to derive the GPSG.

After exploring a handful of approaches to Natural Language Generation both in general as well as in context to Computational Humor, we chose an open source 'Realization Engine' by the name of SimpleNLG to provide the underlying Natural Language Generation capabilities to our prototype.

SimpleNLG is a JAVA library that allows a programmer low-level access to the components of natural language, starting from a lexicon of general English terms and their properties, then defining the inflections (tense, voice, mood etc.) of the desired content, and finally running the data through the realization engine to get a grammatically correct phrase (Gatt and Reiter 91).

Multiple types of phrases further ease the generation process, and simpler types like Noun and Adjective phrases can be stitched together to create Sentence phrases. In addition, the utility of Pre-modifiers and Post-modifiers allows for desired ordering of phrases. Sentence Phrases can further be compounded to generate multi-paragraph text in under a second. We shall use this library and it's provided realization engine as the foundation for our program.

Representation

We present the second research sub-question we aim to answer:

Given a knowledge base and a subject X, can we query the knowledge base to detect a relationship between X and Y (where Y is a numerical quantity)?

If yes, can we make our agent generate a Harper's Index describing the relationship between X and Y following the earlier hypothesized grammar?

To answer this research question, we must first figure out how we will define the 'knowledge base' of facts that we want to query for information. Then we have to figure out how to connect said knowledge base to our text generating JAVA program. Once we do this, then all that remains is running queries for a given subject, and isolating the data points needed to fill in the blanks in our earlier hypothesized GPSG.

For defining our knowledge base, we chose to go with the simplest option available: Prolog. Prolog is a general-purpose first-order programming language that is typically associated with Artificial Intelligence and Computational Linguistics (Lloyd). In Prolog, all facts in the knowledge base are declared in the form of relationships:

predicate(X,Y).

The above format implies one of the following:

1) *X is related to Y*
    a. X is the subject and Y is an object
    b. The predicate is a description of their logical relationship

OR

2) *Y is an attribute of X*
    a. X is the subject and Y is a descriptor
    b. The predicate details what is being described about X

The net result of the set of Prolog statements defines what the universe looks like, and the only logical conclusions that can be made about that universe are that the declared set of objects

exist in it and those objects are related to each other by the relationships declared in it. Nothing more and nothing less exists in this universe.

Let us look at an extremely basic example of what the universe looks like in Prolog, and explore how we declare facts and run queries in this universe.

Table 1: Example of a Knowledge Base

Knowledge Base	**Queries**	**Results**
son(*Ben*, *Jake*)	1) son(*Ben*, *Jake*). 2) son(*Ben*, *Harry*). 3) son(X, *Jake*). 4) Son(Y, *Ben*).	1) **True** 2) **False** 3) X = Ben 4) **False**

In Table 1, the first column is our knowledge base. We declare that a person named *Ben* and a person named *Jack* exist and that *Jake* is *Ben's* son. We attempt a number of queries on this knowledge base (in the second column), the results of which are in the third column.

1) We ask if *Jake* is *Ben's* son. The response is **'True'** as this is correct.
2) We ask if *Harry* is *Ben's* son. The response is **'False'** as no fact or rule exists in this knowledge base to address this query.
3) We ask the computer to tell us who *Jake's* father is. The result is *'Ben'*.
4) We ask the computer to tell us who *Ben's* father is. The result is **'False'** as no fact or rule exists in this knowledge base to address this query.

We shall attempt to run queries on a simple Prolog knowledge base for a given subject, and when the result happens to describe a quantitative fact about the subject, we shall use our earlier JAVA program to generate a Harper's Index about the subject.

For our JAVA process to connect to Prolog, we initially attempted to use open source libraries available online that claimed to allow the definition of Prolog facts using JAVA objects. This theoretically would bring all our work under one JAVA program. However, we were unable to do so because of missing documentation and/or codebases for these libraries, as most of them had become obsolete.

Instead, we had to use a JAVA class called 'ProcessBuilder' which allows a JAVA program to spawn a child process that runs normally but with all Input and Output of the child process redirected to the parent JAVA process. This allows us to spawn a Prolog program and query a pre-defined knowledge base, and then get the results of those queries as variables within the JAVA program.

Once this is done, we will make the user enter Prolog queries and check the data type of the returned value. Anytime the returned value is a detected to be a numeral, then the program will enter the generation part of the program, and will attempt to generate a Harper's Index about the subject using the GPSG we will have derived earlier.

CHAPTER III

RESULTS

We present to you the results of the two halves of the research project, and shall then explore what the combined result looks like.

Generation

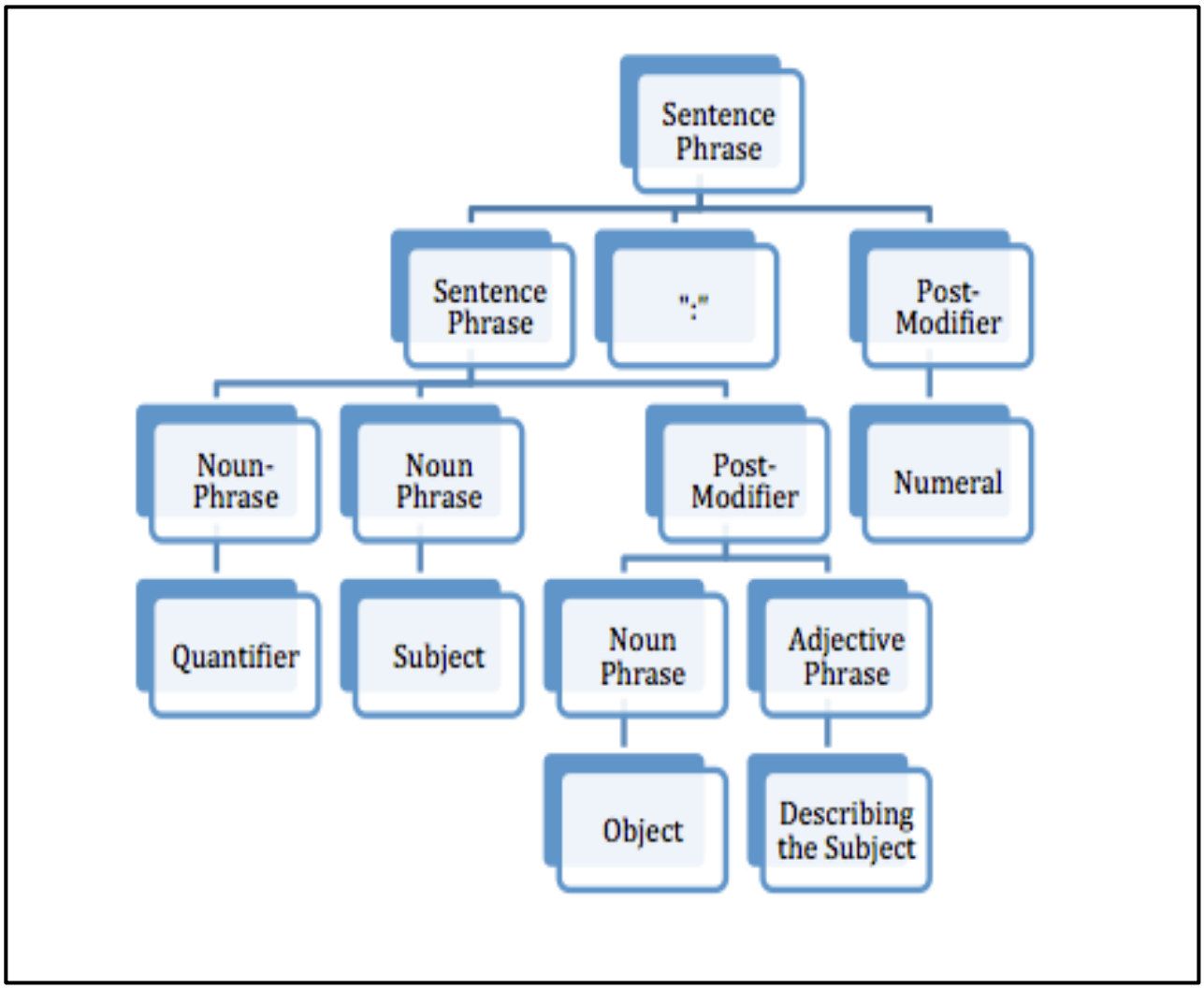

Figure 2: The hypothesized grammar for the average computer generated Harper's Index.

Following the logic presented in Figure 2, here is what the process for generating a Harper's index looks like:

1) Declare the quantifier being used within a Noun Phrase
2) Initialize the subject within a Noun Phrase
3) Create a Noun phrase for an object OR a Adjective Phrase for a descriptor
4) Define the inflections of an empty Subject Phrase
5) Under this Sentence Phrase:
    a. Set the Sentence Phrase to be equal to the Subject Noun Phrase
    b. Set the Quantifier Noun Phrase as its Pre-Modifier
    c. Set the object/descriptor phrase its Post-Modifier
6) Add a colon as a Post-Modifier to the Sentence Phrase at the end of step 5.
7) Add the numeral quantity as a Post-Modifer to the Sentence Phrase at the end of step 6.
8) Pass this Sentence Phrase into the SimpleNLG Realization Engine.

The result is a finished computer generated Harper's Index. This leads us further to the conclusion that all we need to generate a Harper's Index on nearly any subject is the following four data points:

1) Quantifier
2) Subject
3) Object OR Descriptor
4) Numeral

Let us look at a few examples of real human-written Harper's Indices, and compare them to the requirements of our grammar.

Example #1

"Number of amendments made to a February parliamentary bill legalizing same-sex marriage in France: 5,395"

Table 2 details the necessary data points needed for the agent to construct the above Harper's Index through a color-coded comparison.

Table 2: Data Comparison for Example #1

Number of amendments made to a February parliamentary bill legalizing same-sex marriage in France: 5395	1) Quantifier 2) Subject 3) Object 4) Numeral

Example #2

"Number of classroom teachers on Forbes Magazine's 2018 30 Under 30 Education list : 0"

Table 3 details the necessary data points needed for the agent to construct the above Harper's Index through a color-coded comparison.

Table 3: Data Comparison for Example #2

Number of classroom teachers on *Forbes Magazine*'s 2018 30 Under 30 Education list: 0	1) Quantifier 2) Subject 3) Object 4) Numeral

Example #3.1

"Percentage of Americans who support White nationalism: 7"

Table 4 details the necessary data points needed for the agent to construct the above Harper's Index through a color-coded comparison.

Table 4: Data Comparison for Example #3.1

Percentage of Americans who support White nationalism: 7	1) Quantifier 2) Subject 3) Object 4) Numeral

Example #3.2

"Percentage of Americans who believe whites in America are under attack: 39"

Table 5 details the necessary data points needed for the agent to construct the above Harper's Index through a color-coded comparison.

Table 5: Data Comparison for Example #3.2

Percentage of Americans who believe whites in America are under attack: 39	1) Quantifier 2) Subject 3) Descriptor 4) Numeral

While we believe that our grammar meets the requirements for the average Harper's Index (having gone through hundreds of examples in the beginning of this study), we acknowledge that there are times when the grammar falls short. Here is an example:

Example #4

"Factor by which women's Tinder messages are longer than men's: 10"

Table 6 details the necessary data points needed for the agent to construct the above Harper's Index through a color-coded comparison.

Table 6: Data Comparison for Example #4

Factor by which women's Tinder messages are longer than men's: 10	1) Quantifier 2) Subject 3) Descriptor 4) Object 5) Numeral

What sets this example apart is that there is neither a simple subject-object relationship, nor a simple subject-descriptor relationship. There is a combination of both, in order to compare a subject and an object. While we can still force our code to generate this example by picking one of the two, it requires some extra work on the part of the programmer. However this is not an issue since most of the time, Harper's Indices that seek to bring out contrast operate in pairs rather than in isolation, so this is an outlier that can be ignored for the sake of our study. Therefore we can stick to our four-part grammar when we proceed to study representation.

Representation

Once we were successful in generating Harper's Indices as shown in the examples above, we needed a way to actually get information about a subject needed for the construction of said index rather than to just provide all the information ourselves. Simply put, we needed to connect a knowledge base querying engine to our Harper's Index generator. As discussed in Section II, we used Prolog to generate a very basic knowledge base. Said engine takes Prolog queries as text inputs from the user, and if it discovers a quantitative data type as the returned value, it generates a Harper's Index. Let's look at some examples of what we were able to generate.

Example #5.1

Say we start with a very basic knowledge base that contains only one fact: that *Jake* is 31 years old. Say the user then queries the database to find *Jake's* age (as shown in Table 7).

Table 7: Knowledge Base for Example #5.1

Knowledge Base	Queries	Results
age(*Jake*, 31).	1) age(*Jake*, X).	1) X = 31

Since 31 is a numeral, the computer generates a Harper's Index. Here's the result:

"Age of Jake: 31"

Obviously, this result is not as verbose and descriptive as some of the earlier examples we have given. It has a subject, quantifier and numeral, but is missing the fourth element needed to make a Harper's Index: an object OR a descriptor. The next example tries to address this as simplistically as possible.

Example #5.2

We tried to combine the results of multiple queries about the same subject (in this case, *Jake*) into a single Harper's Index. To do this, let us add one more fact to the knowledge base; that *Jake* is *Ben's* son (as shown in Table 8).

Table 8: Knowledge Base for Example #5.2

Knowledge Base	Queries	Results
son(*Ben*, *Jake*). age(*Jake*, 31).	1) son(X, *Jake*). 2) age(*Jake*, Y).	1) X = *Ben* 2) Y = 31

We stored X (*Ben*) in the hopes of using it later, and then started the Harper Index generation process when we detected that the variable Y had a numerical quantity (31) stored in it. Here's the net result:

"Age of Jake who is Ben's son: 31"

This example has all four data points needed to meet the parameters of the earlier derived GPSG. This shows that theoretically, all we need for more complicated computer generated Harper's Indices is to combine the results of multiple queries as long as the subject is constant.

Example #6

Table 9: Knowledge Base for Example #6

Knowledge Base	Queries	Results
daily_payment_Arkansas(Jury duty, $50).	daily_Payment_Arkansas(Jury duty, X).	X = $50

We could say that the Harper's Index generated from the knowledge base in Table 9 is:

"Dollars in daily payment in Arkansas for jury duty: $50"

However, this example shows the limitations of the Prolog language for defining logical relationships, in so far as their application to Natural Language Generation is concerned. While Prolog works fine for behind the scenes work of using predicates as descriptors of a logical relationship, they force the programmer to do the busy work of splitting up terms like ‘daily_payment_Arkansas’ to make them usable in a sentence. This is not ideal, and shows that future researchers should consider alternatives to Prolog that offer a more verbose method of defining logical relationships.

Net Result

The above examples show that with the combined effort of the agent (which recognizes the four fundamental data points) and the programmer (who defines the inflections for grammatical correctness), our prototype can generate a pretty readable Harper’s Index. Our software prototype thus opens a continuously running process that keeps asking the user for PROLOG queries. It runs each query, and each time the result of a certain query is a numerical quantity, it knows to detect the subject of the query and the queried function itself to attempt to generate a Harper’s Index.

CHAPTER IV

CONCLUSION

What we've accomplished

We have accomplished the following:

1) We can produce more complicated Harper's indices if we hardcode our subject, descriptors and quantity data.
2) We can retrieve and isolate quantifiable data from a knowledge base.
3) We can use that data to produce a very simple Harper's index.

What's next?

1) Can we combine results of queries to create more descriptive phrases?

   We showed in Example 5.2 of Section III that we could indeed perform multiple queries to generate more descriptive Harper's Indices. However, for more realistic examples, the object and the numeral parts of the Index have to actually be related to each other. Therefore future research will need to study ways to query for multiple attributes of a subject where those attributes chosen for the Index complement each other in the story they tell about the subject.

2) Can we add contrast?

   Numerous examples of Harper's Indices exist in pairs of contrasting information, as contrast is the easiest way to force someone to question whether the state of affairs of the given subject is appropriate. Currently, each Index in such a contrast-driven pair has to be separately generated. Therefore future research will have to find ways to detect contrasting numerical data points within a knowledge

base, and be able to generate a pair of Harper's indices when it detects this. The question is one of whether this happens on the Representation side, where a special type of fact is created to acknowledge contrast; or on the Generation side, where the query is crafted in a way that it has some method to detect contrast and generate a pair rather than a single Harper's Index accordingly.

Impact on the field of Computational Humor

It is obvious to anyone reading this thesis that the average Harper's Index is not a joke; so much as it is a commentary on our current state of affairs. Its choice as the medium to explore humor was not made by mistake; it was by design. All prior successful research in the field has started from studying 'What is funny?' and therefore starting from a Representation problem. This means that by the time researchers have gotten to the Generation part of the problem, they are severely restricted in the domain of topics (Stock and Strapparava 60). This is why researchers in the field have struggled to go beyond initial successes in puns and knock-knock jokes, limited to the kind of humor "in general circulation among school children" (Binsted and Ritchie 25). We wanted to take advantage of a fill in the blanks approach starting from the Generation side, which gave us far more room for varying grammatical inflection. This accommodated for a greater variety of subjects than has been earlier possible, leading to a robust grammar, which can easily be adapted for all kinds of NLG, including humor. This follows of the idea that "It might be beneficial to look at the development of humor theory and possible applications that don't require a general theory of humor; this might be the only way to bring the field forward" (Nijholt 64). It is our hope that future researchers can use our success on the Generation front to link up more complicated Representation engines built to detect relationships that are actually 'funny'.

REFERENCES

Gatt, Albert and Ehud Reiter. "SimpleNLG: a realization engine for practical applications", *Proceedings of the 12th European Workshop on Natural Language Generation*, March 30-31, 2009, pp.90–93

Binsted, Kim and Graeme Ritchie. "Computational Rules for Punning Riddles", *International Journal of Humor Research* 10.1, 1997, pp. 25–76.

Stock, Oliviero and Carlo Strapparava. "Getting Serious about the Development of Computational Humour" *Proc. 8th International Joint Conference on Artificial Intelligence* (IJCAI 03), Morgan Kaufmann, 2003, pp. 59–64.

Nijholt, Anton. "Embodied Conversational Agents: 'A Little Humor Too.'" *IEEE Intelligent Systems* 21.2, 2006, pp. 62–64.

Lloyd, John W. *Foundations of Logic Programming.* Berlin: Springer Berlin, 2013. Print.